Initial State Prediction in Planning

Michael Cashmore and **Bram Ridder**
firstname.lastname@kcl.ac.uk
King's College London

Senka Krivic and **Justus Piater**
firstname.lastname@uibk.ac.at
University of Innsbruck

Abstract

Offline reasoning techniques and online execution strategies have made planning under uncertainty more robust. However, the application of plans in partially-known environments is still a difficult and important topic. In this paper we describe our work using Maximum-Margin Multi-Valued Regression to predict new information about a partially-known initial state, represented as a multigraph of relations. We evaluate this approach in a robotics domain, demonstrating high recall and accuracy, leading to more robust plans.

1 Introduction

Planning in the domain of robotics means planning with incomplete and uncertain information. In such domains plans generated can be fragile. Contingency planning with belief states [Bonet and Geffner, 2000; Hoffmann and Brafman, 2005], conformant planning [Smith and Weld, 1998; Palacios and Geffner, 2006], and replanning techniques [Brafman and Shani, 2014] work to make execution more robust, for an acceptable cost of computational difficulty or plan quality.

In this paper we focus on the problem of planning with incomplete information in a deterministic domain. We describe a preprocessing step that predicts new information about a partially-known initial state represented as a multigraph of relations. We pose the problem of prediction as that of learning missing edges in a graph. The learned edges are analogous to assumptions about facts within the initial state. The problem is similar to a restricted class of contingency planning problems with deterministic actions, *deterministic POMDPs* [Bonet, 2009].

To solve our prediction problem we use a kernel-based approach to learn missing edges in a partially-given multigraph [Krivic et al., 2015]. This involves propagating knowledge from existing to unknown relations.

The method is a version of Maximum Margin Multi-Valued Regression (M^3VM) [Ghazanfar, Prügel-Bennett, and Szedmak, 2012; Szedmak, Ugur, and Piater, 2014] extended to the class of learning problems where item-item relations might be given by different attributes. This learning framework was used at the core of a recommender system [Ghazanfar, Prügel-Bennett, and Szedmak, 2012] and for predicting the effects of an action on pairs of objects in an affordance learning problem [Ghazanfar, Prügel-Bennett, and Szedmak, 2012]. They have shown that it can deal with sparse, incomplete, and noisy information.

Krivic et al. [2015] use this method to refine a world model for planning to tidy up a child's room with a robotic agent. We build on this, describing how the approach can be generalised for use in any planning domain, learning edges that correspond to generic propositions in the initial state. In particular we describe: how the initial state is represented as a partially-known multigraph; the kernel-based approach to learning missing edges; how the results of the learning tool are translated back into the planning domain; and finally, how they can be used in the planning process.

We demonstrate its efficacy on an extension of the tidy-room domain proposed by Krivic et al. performing an empirical evaluation on a range of problems.

In Section 2 we describe our problem formulation for learning new relations in partially-known initial states. We describe the learning tool in Section 3, and explain how the learning tool is used to solve the formulated problem. In Section 4 we perform an evaluation. We conclude in Section 5

2 Predicting in the Planning Problem

In this section we describe in detail our preprocessing step, which predicts new information about a partially-known initial state.

Definition 1 (Planning Problem) *A planning instance* Π *is a pair* $\langle Dom, Prob\rangle$*, where* $Dom = \langle Ps, As, arity\rangle$ *is a tuple consisting of a finite set of predicate symbols* Ps*, a finite set of (durative) actions As, and a function arity mapping all symbols in* Ps *to their respective arities. The triple* $Prob = \langle Ob, Init, G\rangle$ *consists of a finite set of domain objects Ob, the partial initial state Init, and the goal specification G.*

The atoms of the planning instance are the (finitely many) expressions formed by applying the predicate symbols Ps to the objects in Os (respecting arities). The resultant expressions are the set of propositions P.

A state s is described by a set of literals formed from the propositions in P, $\{l_p, \neg l_p, \forall p \in P\}$. If every proposition from P is represented by a literal in the state, then we say that s is a *complete state*. A *partial state* is a set of literals $s' \subset s$, where s is a complete state.

The initial state $init$ is a partial state. A partial state can be *extended* into a complete state.

Definition 2 (Extending a Partial State) *Let s' be a partial state of planning problem Π. Extending the state s' is a function $Extend(\Pi, s') : s' \rightarrow s$ where s is a complete state and $s' \subset s$.*

We describe a pre-processing step implementing $Extend$. All unknown propositional values in a partially-known initial state are predicted, producing a complete initial state. Briefly, the function $Extend(\Pi, s')$ is implemented as follows: the initial state $init$ is converted into a multigraph; edges in the multigraph are learned using a relational learner; then the new edges are added as literals to the initial state.

First we describe the construction of the multigraph, then in Section 3 we describe the relational learner, and then how the learned relations are inserted back into the initial state. Finally, the complete initial state can be used by a classical planner to generate a plan.

Constructing the Multigraph

We represent a partially-known initial state $init$ as a partially-known multigraph M.

Definition 3 (Partially-known Multigraph) *A partially-known Multigraph M is a pair $\langle V, E' \rangle$, where V is a set of vertices, and E' a set of labelled, directed edges.*

We use E' denote a set of edges in a partially-known multigraph, while a complete set of edges is E. The partial state $init$ is described as a multigraph with edges for any proposition that is unknown, or known to be true. That is:

$$V \leftrightarrow Ob$$
$$E' := \{e, \forall p | \neg l_p \notin init\}$$

The existence of a *directed edge* between two vertices for a predicate $pred$ is described by the function $L_{pred} : V \times V \rightarrow \{0, 1, ?\}$. For example, let b and u be two vertices in set V. For proposition p involving objects b and u, $L_{pred}(b, u) = 0$ if $\neg l_p \in init$, $L_{pred}(b, u) = 1$ if $l_p \in init$, and $L_{pred}(b, u) =?$ otherwise. Edges are directed in the parameter order from the domain.

Example

Consider the planning problem in figures 1 and 2. The problem describes a robot that is able to move between waypoints, pickup and manipulate objects, and put them in boxes. The predicates: `pickup`, `push`, `stack-on`, `fit-inside` describe whether it is possible to perform certain actions upon objects in the environment. In this problem we restrict our attention to three objects: `cup01`, `box01`, and `block01`. In general literals that are absent from the initial state are assumed to be false. However, in this initial state those literals are assumed to be unknown. It is also possible in PDDL2.1 to specify propositions that are initially false using statement `(not p)`, in which case $L_{pred}(b, u) = 0$.

A graph M is generated, the vertices of which are $Ob := \{rob, cup01, box01, block01\}$. The graph is shown in figure 3.

```
(define (domain toy-domain)
(:requirements :strips :typing ...)
(:types
  waypoint
  vehicle
  interactable
  box toy - interactable
  block - toy
  gripper)

(:predicates
  (pickup ?r ?interactable)
  (push ?r ?interactable)
  (stack-on ?interactable ?interactable)
  (fit-inside ?interactable ?box)
  ...)

(:durative-action goto ...
(:durative-action pickup ...
(:durative-action putdown ...
(:durative-action stack ...
(:durative-action unstack ...
(:durative-action put_in_box ...
)
```

Figure 1: A fragment of the `toy-domain`. Some predicates and the body of operators are omitted for space.

```
(define (problem toy-example-problem)
(:domain toy-domain)
(:objects
  wp1 wp2 wp3 ...  - waypoint
  robot - vehicle
  cup01 ... - interactable
  box01 ... - box
  block01 ... - block
)
(:init
  (pickup robot cup01)
  (push robot block01)
  (push robot cup01)
  (stack-on box01 block01)
  (stack-on block01 cup01)
  ...)
(:goal ...)
)
```

Figure 2: A fragment of an example problem from the `toy-domain`. Some objects, part of the initial state, and the goal are omitted for both space and readability. In this example problem we restrict our attention to three objects (`cup01`, `box01`, and `block01`) and the five propositions over these objects that are known in the partial initial state.

3 Predicting Missing Edges in a Multigraph

In this section we describe the procedure of predicting missing edges in a partially-known multigraph. We are using (M^3VM) framework to extract similarities among nodes based on known edges and to estimate missing edges.

The goal is to predict directed edges between all vertices in the same set V. Generalized, this is a problem of predicting connections from the vertices in a set B to the vertices in a set U. In our problem $B = U = V$. In the following we use B and U to differentiate between vertices of outgoing and incoming edges respectively. Edges between the vertices are describing a relation between the sets B and U. In order to capture structure of this relation we reconstruct a function $F : B \times U \rightarrow E$ from knowledge on existing edges. The function F is describing the mapping of vertices to a complete

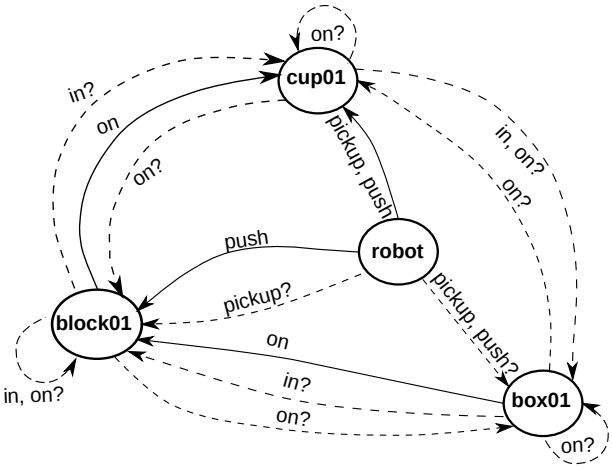

Figure 3: The graph M representing the initial state in the example problem. Solid edges correspond to propositions known to be true, $L_{pred}(b, u) = 1$. Dashed edges correspond to propositions whose value is unknown in the initial state, $L_{pred}(b, u) =?$. In this figure, for space, we use *in*, and *on* in place of *fit-inside*, and *stack-on* respectively.

set of edges E.

The function F is indirectly and partially given by the subset E'. Reconstructing F can be done by finding a vector valued function that extends E' to all possible pairs of vertices.

For vertices b and u, we define the vector

$$\mathbf{e_{bu}} = \{L_{pred}(b, u), \forall pred\}$$

For example, in the graph given in figure 3 there will be the vector:

$$\begin{aligned}&\mathbf{e}_{robot,block01}\\ &= [L_{fit-inside}(robot, block01),\\ &\quad L_{stack-on}(robot, block01),\\ &\quad L_{push}(robot, block01),\\ &\quad L_{pickup}(robot, block01)]\\ &= [0, 0, 1, ?]\end{aligned}$$

Then, we define the projections of known edges E' into a set containing start vertices B and end vertices U by

$$\begin{aligned}B' &= \{b \in B | L_{pred}(b, u) \neq ?, \forall pred, \forall u \in U\}\\ U' &= \{u \in U | L_{pred}(b, u) \neq ?, \forall pred, \forall b \in B\}\end{aligned}$$

Finally, the learning problem is given by a set of sample items consisting of three elements $(b, u, \mathbf{e}_{bu})$ where: $(b, u) \in B' \times U'$, and $\mathbf{e}_{bu} \in E'$. To realize this learning task we set up an optimization routine for maximum margin regression.

To express the connection between the known instances and their subsets we assume that:

Condition 1 *There exists a mapping ϕ from V into a Hilbert space H_ϕ, with the kernel function $\kappa_{vertice}$ defined on all possible pairs B' and U' of all subsets of $B \times U$ such that $\kappa_{vertice}(B', U') = \langle \phi(B'), \phi(U') \rangle$.*

Condition 2 *Similarly there is another mapping ψ of E into a Hilbert space H_ψ equipped by a kernel function κ_{edge} defined for all pairs $e_1, e_2 \in E$ such that $\kappa_{edge}(e_1, e_2) = \langle \psi(e_1), \psi(e_2) \rangle$.*

Hilbert spaces H_ϕ and H_ψ are feature representations of the domains of B, U, and E. This allows us to use the inner product as a measure of similarity. A rigorous description of the mapping into Hilbert spaces, and the construction of the optimisation problem can be found in Krivic et al. [2015].

After mapping, and solving the joint optimization problems that we have constructed, have for all $b \in B$

$$\mathbf{W}_b = \sum_{u \in B'} \alpha_{bu}(\psi(\mathbf{e}_{bu}) \otimes \phi(u)) \tag{1}$$

where (α_{bu}) are the optimal Lagrange multipliers and $\mathbf{W}_b$ is a linear vector used to produce a prediction over a given edge. The prediction to a given edge $e(\hat{b}, \hat{u}) \in E/E'$ can be derived by

$$e^*_{\hat{b},\hat{u}} = \max_{\hat{b}u \in E'} \psi(\mathbf{e}_{\hat{b}\hat{u}}) \mathbf{W}_{\hat{b}} \phi(\hat{u})$$

We can expand $\mathbf{W}_{\hat{b}} \phi(\hat{u})$ using (1) to obtain:

$$\sum_{u \in U'} \alpha_{\hat{b}u} \underbrace{\langle \psi(e_{\hat{b}\hat{u}}), \psi(e_{\hat{b}u}) \rangle}_{\kappa_{edge}(e_{\hat{b}\hat{u}}, e_{\hat{b}u})} \underbrace{\langle \phi(u), \phi(\hat{u}) \rangle}_{\kappa_{vertice}(u, \hat{u})}$$

where $\kappa_{edge}(e_{\hat{b}\hat{u}}, e_{\hat{b}u})$ and $\kappa_{vertice}(u, \hat{u})$ are the kernel matrices built on the inner product between the corresponding elements.

For each prediction, $e*_{bu}$, we update the existence of the directed edges:

$$L_{pred}(b, u) = e*_{bu}$$

Thus, the graph is completed.

Extending a Partial State with Predictions

Given a complete multigraph, we extend the partially-known initial state $init$ by adding literals to the state:

$$(l_p \in init) \leftrightarrow (L_{pred}(b, u) = 1)$$

where l_p is the positive literal of proposition p, formed from predicate $pred$ between objects b and u. For example, figure 3 contains an edge representing the proposition `pickup(robot, block01)`.

Initially, $L_{pickup}(robot, block01) =?$. After prediction, $L_{pickup}(robot, block01) = 1$ Therefore, we add to the initial state the literal

(pickup robot block01)

4 Evaluation

To evaluate the approach, we generated randomised problem instances of the example domain, varying the initial knowledge between 1% and 95%, in 1% increments. This was done by removing literals at random. For each increment five initial states were generated, and the results were averaged. In every case the learning process took less than two seconds.

Figure 4 illustrates results for problems with 80 objects. In this graph we separate propositions that are between two objects (`stack-on`, `fit-inside`) and propositions between the agent and the object (`pickup`, `push`).

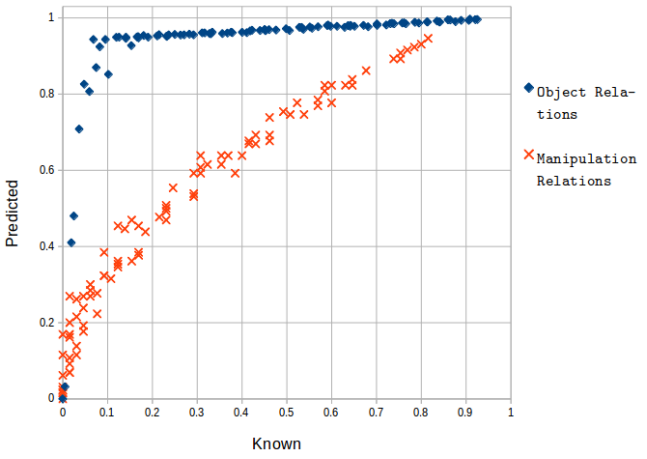

Figure 4: Prediction as fraction of the total real propositions, against initial knowledge. As the initial knowledge is increased (left to right) the number of predicted edges increases. Results are separated into propostions between two objects (`stack-on`, `fit-inside`) and relations between the robot and the object (`pickup`, `push`).

The number of learned relations is large: for a small amount of initial knowledge (30%) the recall of the process is almost 90%. Furthermore, with 80 objects, there were no propositions whose values were predicted with error.

In order to determine a minimum number of objects required for learning meaningful edges, we generated problems with 8, 16, 32, and 72 objects (still with a single robot) in the same way. The number of learned relations are shown in figure 5.

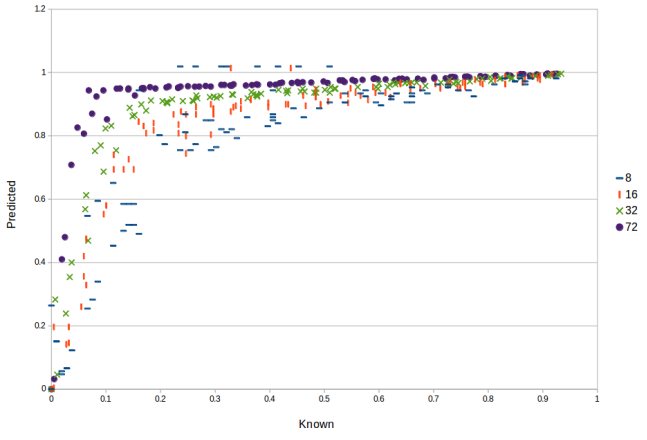

Figure 5: Prediction evaluated as fraction of total real propositions against initial knowledge. As the initial knowledge is increased (left to right) the number of predicted edges increases. Results are shown for instances with varying numbers of objects ($|Ob| = 8, 16, 32, 72$). Data points for which predicted edges are greater than 1 indicate that propositions were predicted erroneously.

Data points for which the predicted fraction is greater than 1 indicate that some erroneous edges were learned. For other data points, there were no errors. These inaccurately predicted propositions appear when there are fewer objects (8 and 16). With 32 objects and greater, the learning approach proved robust. In general, a greater number of objects, and therefore known relations between them, will provide a better source for the learner.

The graphs show that in each set, as the initial knowledge increases the initial state is improved dramatically. With only 8 objects and 30% of the initial state, 60% of the complete initial state can be retrieved. Below 30%, performance drops off dramatically.

This combination of accuracy and recall allows us to solve many otherwise unsolvable instances, while maintaining a high degree of robustness. Table 1 shows the number of valid plans generated for problems (with 80 objects) after preprocessing. Without preprocessing, no problems could be solved.

We used an optimistic approach to solve the same problems. As this approach suggests, unknown relations were assumed to be true. This means that the planner expects all objects of the same PDDL type to possess the same qualities, and that the executive can manipulate these objects in full. This comparison shows a naive approach to completing a partially known initial state, which often produces invalid plans. The purpose of this comparison is to show a base improvement in robustness against a pure replanning approach.

Initial knowledge	10	15	20	25	30	35	40	45
Optimistic	2	1	2	3	2	5	3	2
Prediction	0	0	0	4	5	5	5	5
Initial	50	55	60	65	70	75	80	90
Optimistic	2	2	5	3	5	3	5	5
Prediction	5	5	5	5	5	5	5	5

Table 1: Number of problems for which valid plans were generated, for varying percentages of initial knowledge. For each column, we tested five problems. Results are shown for initial states after prediction, and initial states after an optimistic prediction. Initial states with no prediction did not produce valid plans.

5 Conclusion

We have shown how an initial state with uncertainty can be represented as a partially-known multigraph, how the M^{3}VM framework can be used to predict edges in such a graph, and how these edges can then be reintroduced into the initial state as predicted propositions.

This approach is performed offline and is not an execution strategy in itself. It is orthogonal to other approaches in dealing with uncertainty in the initial state, and can be combined. Without integration into a more sophisticated execution strategy, our evaluation has shown that this approach accurately predicts a surprisingly large number of facts in a structured domain, even with few objects.

References

[Bonet and Geffner, 2000] Bonet, B., and Geffner, H. 2000. Planning with incomplete information as heuristic search in belief space. In *Proceedings of the 5th International Conference on Artificial Intelligence Planning Systems (AIPS'00)*, 52–61.

[Bonet, 2009] Bonet, B. 2009. Deterministic POMDPs revisited. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI'09)*, 5966.

[Brafman and Shani, 2014] Brafman, R. I., and Shani, G. 2014. Replanning in domains with partial information and sensing actions. *CoRR*.

[Ghazanfar, Prügel-Bennett, and Szedmak, 2012] Ghazanfar, M. A.; Prügel-Bennett, A.; and Szedmak, S. 2012. Kernel-mapping recommender system algorithms. *Information Sciences* 208:81–104.

[Hoffmann and Brafman, 2005] Hoffmann, J., and Brafman, R. I. 2005. Contingent planning via heuristic forward search witn implicit belief states. In *Proceedings of the 15th International Conference on AutomatedPlanning and Scheduling (ICAPS'05)*, 71–80.

[Krivic et al., 2015] Krivic, S.; Szedmak, S.; Xiong, H.; and Piater, J. 2015. Learning missing edges via kernels in partially-known graphs. In *European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*.

[Palacios and Geffner, 2006] Palacios, H., and Geffner, H. 2006. Compiling uncertainty away: Solving conformant planning problems using a classical planner (sometimes). In *Proceedings of the 21st Conference on Artificial Intelligence (AAAI'06)*.

[Smith and Weld, 1998] Smith, D. E., and Weld, D. S. 1998. Conformant graphplan. In *Paper presented at the meeting of the AAAI/IAAI (AAAI'98)*, 889–896.

[Szedmak, Ugur, and Piater, 2014] Szedmak, S.; Ugur, E.; and Piater, J. 2014. Knowledge propagation and relation learning for predicting action effects. In *Intelligent Robots and Systems (IROS'14)*, 623–629.